User Manual to Inverted Index Visualizer

Maximilian Sohrt

08.01.2009

Contents

1 Programm Purpose **4**
1.1 About Inverted Index 4
1.2 Inverted Index Function 4
1.3 Visualization Purpose 5

2 General usage **5**
2.1 Starting the programm 5
2.2 Using the Visualisation 6
2.2.1 Last Arrangements 6
2.2.2 The GUI Explained 6
2.2.3 Getting Info on a Single Token 8

3 Customizing the programm **10**
3.1 Plugin Interface 10
3.2 First steps 10
3.3 IDocument - Container for document information 11
3.4 Customizing the Index 11
3.5 Getting your terms individually - the IndexWriter 12
3.6 Find what you are looking for - the Search 12
3.7 Summary of all plugins and functions 13
3.8 Final Notes 13

4 Additional Information **14**
4.1 The complete SQL-Structure 14

This manual is meant to help users of our software "Inverted Index Visualizer" to understand it and, for the more advanced user, to extend it for his or her personal needs. Beside the Javadoc this document should be a help to understand the source code and extend individually.

1 Programm Purpose

1.1 About Inverted Index

An Inverted Index is an index data structure to store information about a document collection, such as its contents, score and a lot of things that can be defined by the user. The purpose is to do fast full text searches with the possibility to define own criteria to sort the result and to show the best matches for the query. It is the most popular way of searching at the moment. The well-known search engines on the WWW like Google, MSN and Yahoo use it for their indexes.
But there are also other applications that use an inverted index. For example the built-in Search in Windows XP has the possibility to build an index of your hard disk contents. Nevertheless there are some not that famous areas of application, such as the searches in many companie's intranets and product searches in several web shops which even allow fault-tolerant and phonetic searches to guide the users to the really interesting results. In some cases the results here can be directly manipulated by the administrator. For example if a shop wants to promote special products which normally would not be highly ranked, the rank can be changed by changing its weight by hand.
With the purpose to give an outlook a relatively new field of application is mentioned here: The search in audio and video files. It is very hard to extract information from these kind of files because spoken language is very different and it is at the moment nearly impossible to get the content out of pictures by analysing its structure. But some of the big search engines on the WWW tried to get at least the spoken content out of the video and audio files on the web. As a result there now is the possibility to search in these files.

1.2 Inverted Index Function

This section is meant to give the reader a short insight on how an inverted index works. First of all the name "Inverted Index" will give you a clue on the functionality. It is inverted because not every document in the collection will be crawled on a search. Also the documents will not be saved with all the words in it. It happens so to say "upside-down".
When inserting a new document the programm will check for every word in that document if it already exists. If not, it will be inserted and the unique ID of the document just added will be connected to that word. It it exists only the ID will be connected to that word. The result is that you have only words in your database that have the containing documents as "children". On first sight this looks like too much work but if your index is getting larger and larger you will have nearly all words of a language in your index and the work is getting much smaller then search through all these documents. So the advantage is huge but only for big document collections.
To give some numbers: You have to imagine that there are about 200.000 to 300.000

words in languages like German or English. And all the words in an index are stemmed, which means grammar is removed and you only have "bare" words.

1.3 Visualization Purpose

Although it might be clear what the purpose of this programm is, you can get some information about the main thoughts behind our programm.
First of all the activities on a often used index are very complex. As a creator or administrator of such an index you have hard work to do if you want to understand and trace these activities. Try to imagine the database behinde google.com for example and then try to optimize it by only having the possibility to see what the web frontend shows you. You cannot see why a certain algorithm sorts in what way. You cannot see all documents connected to a special word and so on.
And here it comes, the visualization. The purpose is to show the huge amount of information directly and without any information you do not need. With this programm you have the chance to show several thousands of items on one screen and to see what happens to them. You can directly see the changes of a new algorithm. You can compare it to old or other algorithms.
Another purpose is to easily extend the programm with the result that it can fit your very own needs. Therefore we created a plugin interface that allows you to do this.

2 General usage

2.1 Starting the programm

If you just want to use the programm you only need Sun's JRE (Java Runtime Environment) which you can download here, if you don't have it installed already: Java download page.
Depending on where the database server with your index is you need a working ethernet connection or even internet connection. Later on the manual shows you, how to choose your database server, no matter where it runs, LAN or Web. If you are going to insert documents into the index, much bandwith is needed. Depending on the documents file size it can take several minutes to insert a document, because you have to transfer nearly the whole file to the database server, after it has been parsed. If you have a 20MB pdf-file with nearly no images, there will be 20MB traffic to get the document into the index. So do not try this through you mobile phone connection unless you have a flatrate for surfing mobile.
If you want to change the source code, you need a java editor of your choice and Sun's most recent JDK (Java Development Kit) which can also be found on the same website mentioned above.
In the main-folder you can find the executable to start the programm. Furthermore you

can find the plugins directory which contains the three parts which are customizable (Index, IndexWriter and Search).
If there are no errors, because of wrong java version or else, you should now see the programm booting up.

2.2 Using the Visualisation

2.2.1 Last Arrangements

This part is divided into several smaller parts. When you got the programm up and working you now have a decision to make. This few questions shall help you:
Do you already have a database on a database server on LAN or Web? If yes advance to the next question. If not, get one (best would be MySQL by now).
Do you have an index on your database? If yes, again you need to go to the next question. If no, the programm will create a blank index, connecting the first time to your database. Make sure the given database is empty!
Now, you already own an index on your database. If it is not the same form, as the standard index is, you need to write an own SQL Plugin. More about this can be found in section 3.3.

2.2.2 The GUI Explained

If all the requirements are fulfilled we are ready to go and use the programm. The programm is mainly devided into five sections:

1. **The Search Bar:** Enter the words you want to test and visualize here. You may enter a single word or a whole phrase. This section is closely connected to the second section, the Output Frame.

2. **The Output Frame:** In this frame you can see the created visulisation from your search query. And here you have the possibility to get to know more about every single document, token and result from your index like weight, where it comes from, how often and much more.
   Furthermore you have three tabs. In each tab you can "save" a search until you overwrite it or close the programm. Just enter the Searchwords in the search panel, press "Search" and choose the next tab. Then repeat this steps. Doing like this you may compare up to three results. You can the tab even change during animation and not just at the end of an animation.

3. **The Animation Controls:** With the pause button you are able to pause the animation at any point, for example if you found something interesting or if you

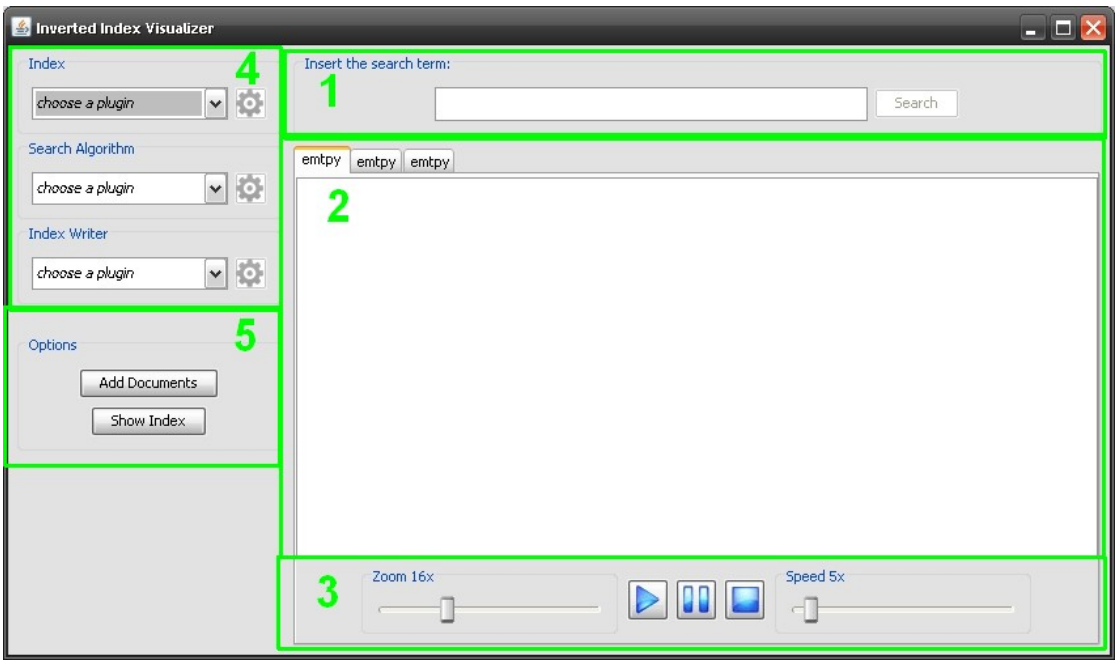


Figure 1: The five sections of the programm

need to get a closer look at a certain state. The play button continues the animation.
The Stop button right to it resets the Output Frame to blank.
The speed controller allows you to change the speed. With speed is meant how fast the objects in the Output Frame move and how fast the intersection algorithms are visualized.
The perhaps most important and interesting is the last control in this section, the zoom. Here you can freely scale the size of the tokens, which means you can have a look at the whole query while processing or if you want to only at a few perhaps very interesting objects.

4. **The Index and Algorithm Chooser:** The first part of this sections allows you to define where index is. To enter your database credentials just push the cogwheel in the index. The dialog that appears then allows you to enter your database credentials. The index dropdown allows you to choose which kind of index you have. Here your very own indexes can be implemented (e.g. distributed index on different servers). More on how to create a plugin for your own database type you can find in section 3.3.
The Search Algorithm Box lets you choose which algorithm you want to use during

your searches. If you push the cogwheel here you will see the stopword list, which can be used to filter words from the beginning and not by weight or other things. Furthermore you can choose in this dialog which intersection algorithm you want to use.
Attention: In the first two dropdown menus you have to choose an algorithm to use the programm.

5. **Add New Documents and Show Index:** In the fifth section of the programm you find the two actions you will not need that often. First of all there is the "Add New Documents" button, which allows you to add new documents to your index. Here you can add nearly all document types, just like doc, xls, ppt, pdf and of course txt and any other plain text format.
   The second button allows you to see the whole Index. This takes quite long because even a small index with about 30 documents has about 5.000 different words and 20.000 connections between words and documents. So just use this button if you have enough time to wait (there will be a status bar, so you can be sure that the programm is not crashed). You will also need lots of RAM if you have a big database.

2.2.3 Getting Info on a Single Token

When the animation has finished after entering some search words, you will have something like this in your output frame:

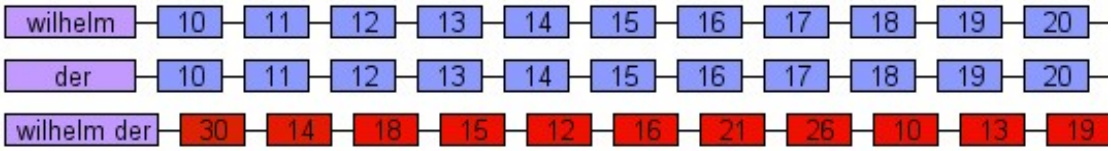


Figure 2: Sample result documents

Now you can get additional information on your single documents. First of all there is the weight of a certain document. Just hover your mouse over the document of interest to get to know its weight and location:

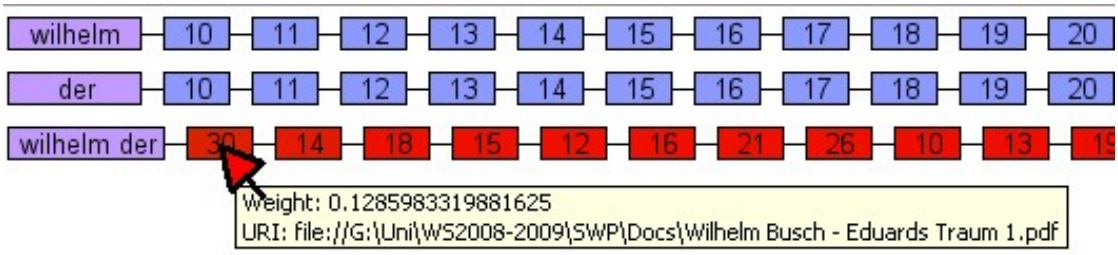
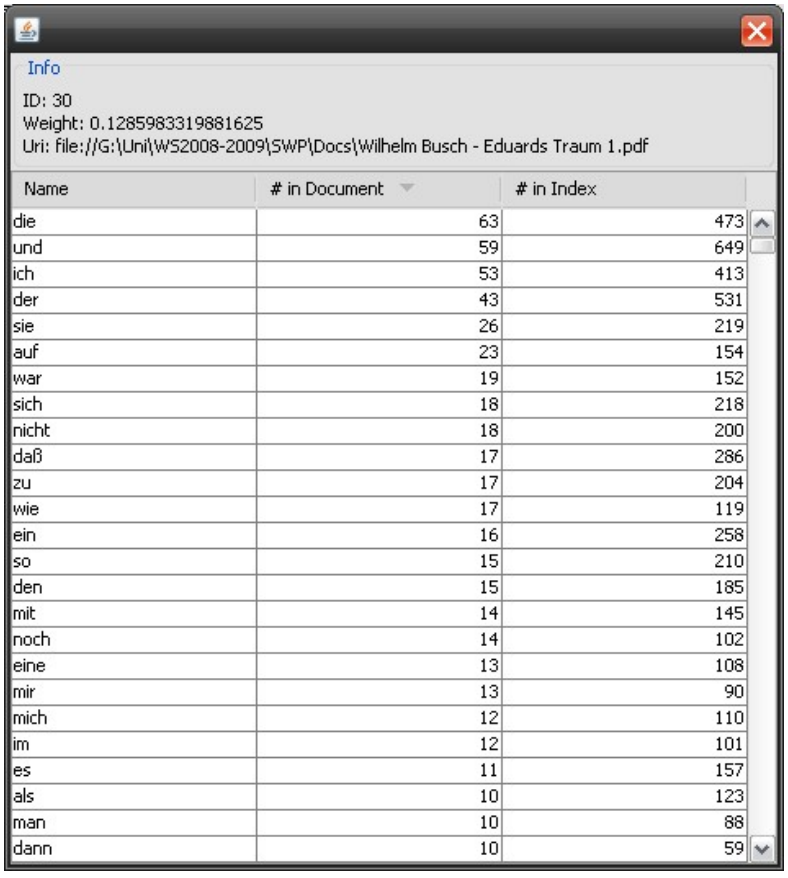


Figure 3: Weight and location of a document

You also can get even more information on a single document if you click on it. You will see the dialog from figure 4.



Name	# in Document	# in Index
die	63	473
und	59	649
ich	53	413
der	43	531
sie	26	219
auf	23	154
war	19	152
sich	18	218
nicht	18	200
daß	17	286
zu	17	204
wie	17	119
ein	16	258
so	15	210
den	15	185
mit	14	145
noch	14	102
eine	13	108
mir	13	90
mich	12	110
im	12	101
es	11	157
als	10	123
man	10	88
dann	10	59

Figure 4: Additional document information

It will show you the unique ID of the document, how often every word in the document appears and directly next to that how often that word appears in the whole index. You can even sort all three columns.
That is the whole thing about the usage of IIV. If you want to customize it, read on.

3 Customizing the programm

3.1 Plugin Interface

The Inverted Index Visualize brings a mighty Plug-In interface which allows you to import your own weighting algorithms, index-structures and search algorithms.
This part of the manual is meant as a help on how to create your own plugins. It will show you the meanings of the functions from the interfaces and give you some hints. It will NOT explain every row of code in our plugins to you and it will not tell you every helper function we used. This would not be helpfull because we want you to have all possibilities when creating your own plugins. Giving too much structure to you would not allow you to do nearly anything you want with the programm and would block the development of own ideas. Furthermore it is the task of the JavaDoc to explain the source code more in detail.
In the following class names will be writen *italic*, names of functions will be written **bold** and package names are written in SMALL CAPITALS.

3.2 First steps

Looking at the folder structure the programm built on your computer will give you a first hint on how to create your own plugins. You will find a folder called "plugins" which contains some more folders. These are "Index", "Indexwriter" and "Search". This allows you to customize any part of the programm with ease. Furthermore the source code will give you further hints.
In general you need to know that the classes for every one of these three plugins have all own interfaces. Just to make it easier, I will list them in table 1. **Important:** Never

Plugin part name	**Interface package and name**	**Class name**
Index	MODELL *IIndex.java*	Index.java
IndexWriter	CONTROLLER *IIndexWriter.java*	IndexWriter.java
Search	CONTROLLER *ISearch.java*	Search.java

Table 1: Plugin classes and their interfaces

change a thing in the interfaces. They are meant to be a guideline on how a new class has to look. If you change something to make your own classes work, every other class,

based on the old interfaces, will not work anymore. Always keep that in mind!
Now we will work through the interfaces and I will shortly explain what every function does.

3.3 IDocument - Container for document information

Beside the plugin we created an own container for information on a document with an own interface. Inside this container values like the unique document id, the uri, the name and the hash are stored. Moreover we implemented some useful functions for this container. Again I use a table to make it easier for you to understand.

This container is explained first, because it needs to be defined in the index. The

Function name	Description
getTermCount	How often does a term occur in the doc
getWeight	Get the weight of document
setWeight	Set the weight
addTerm	Connects the terms with the doc
getTermInfos	Get infos like number of occurences

Table 2: Functions of IDocument

container is also used to get statistics of a single document.

3.4 Customizing the Index

The Index is mainly the part of the programm which connects to your database and fills the index. It does NOT extract the terms out of documents. That is work for the IndexWriter, which is explained in section 3.4.
This gives you the chance to implement your own database. To do so it is the best to start with the already given Index and then customize it.
The first function you will find in IInterface is **createDocument(String uri, String hash)**. This function inserts a new document into your index and returns a handler for the inserted document. The uri is the uri of your document and hash allows you to insert some kind of hash (e.g. MD5) for this document. There is also a version of this function with the additional parameter "name" which is not used by now, but may be used if you also want to insert a name for the document (e.g. the title).
The next function is **getDocumentsByTerm(String term)** gives you all documents that contain the term in the parameter. Return value is a list of IDocument.
The function **getTermCount(String term)** returns the number of occurences of the given term. If you do not hand over a term it will tell you the count of all terms in the collection.

The next two functions are your chance to add your own configuration dialog to your plugin. **hasConfigDialog()** simply returns boolean. True if you want a config dialog or false if you do not want it. **openConfigDialog(JFrame frame)** finally allows to open the dialog in the given JFrame.
The function **getTerms(int order, int limit)** returns a StringList with all terms in the collection. The parameter order allows you to order this StringList and limit allows you to choose a maximum number of terms returned if you do not want to block the computer too long.
setRecorderPanel(IRecorderPanel recorderP) allows you to choose where the recorder panel is, if you want to control the functions of the Index by hand. But regularly it is not needed.
The last function in this interface is **setOwnerFrame(JFrame ownerFrame)** where the parameter is needed to let the programm know in what frame to display the Index information.
This is only a part of what is needed to do. You should and will use some helper functions. This is necessary if you want to do a clean job. If you have no idea what kind of helper functions you may need, just take a look at our example plugin. There you will find some examples such as a function that establishes a connection to you database or else.

3.5 Getting your terms individually - the IndexWriter

This part is also very interesting to customize. It turns the given documents into stemmed and corrected terms. Here you can implement different stemmers for different languages and different kinds of analyzers. But as before the interface just gives you a rough structure while a lot of helper functions are needed to be written by you. Only two functions need to be explained. The rest is like in the Interface of Index.
The first is **addFile(File file)** which is thought to be used to add new files to the index. Normally this function is used to take a file from local storage or the web, normalize it, tokenize it and then send it to your index. As you see this a good place for helper functions. The function **setIndex(IIndex index)** allows you to handle several indexes created through IIndex. That's it. The rest of the functions in this interface is explained above.

3.6 Find what you are looking for - the Search

This plugin hands over the query to your database and returns the result to work with it. This means you only need two functions, one that takes your query and processes it and one that returns the results of that function. The function **setQuery(String query)** sets the search query and brings it into a usable form for the database. After querying the database the results are return by **getSearchResults()** in a List of IDocu-

ments which contains all matching results plus the additional information. The function **getPluginName()** only sets the name of the plugin.
Furthermore you may implement functions here that allow you to intersect the results if you have more then one query word (just to give an example).

3.7 Summary of all plugins and functions

At the end I will give you a fast survey of all interfaces and their functions. This will help you at the beginning of a development process for new plugins until you are familiar with every function. Have fun with this table.

Interface	Function Name	Description
All	**setRecorderPanel**	Control for functions
	setOwnerFrame	Specify target frame
	hasConfigDialog	ConfigDialog yes or no
	openConfigDialog	Define the ConfigDialog
Index	**createDocument**	New Document in index
	createDocument	Alternative (name the document)
	getDocumentsByTerm	Get Docs for a term
	getTermCount	Get Count of a term
	getTerms	Return all terms
IndexWriter	**addFile**	File to parse
	setIndex	Select working Index
Search	**setIndex**	Select working Index
	setQuery	Hand over the query
	getSearchResults	Get the results of query
	getPluginName	Set the name of plugin

Table 3: Review of Plugins

3.8 Final Notes

If you now ask yourself where new documents are connected to already existing terms in the database or added if they do not already exist, I am going to explain this now.

When adding new documents the extracted terms are inserted into the DB. There a TRIGGER checks if the term already exists and appends the document or creates a new term. You can see this in section 4.1 in lines 95-109. We chose that way because it minimizes the traffic between the client programm and the database. So the work is much faster. Furthermore you do not have to build complex SQL structures when inserting a new document.
When creating an instance of IDocument the corresponding terms can be fetched by a function easily from the database.
So the connection is explicit and easy.

4 Additional Information

4.1 The complete SQL-Structure

```
0   -- phpMyAdmin SQL Dump
1   -- version 2.11.1
2   -- http://www.phpmyadmin.net
3   --
4   -- Host: localhost
5   -- Erstellungszeit: 11. Januar 2009 um 03:29
6   -- Server Version: 5.0.45
7   -- PHP-Version: 5.2.4
8
9   SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
10
11  --
12  -- Datenbank: `swp`
13  --
14
15  -- --------------------------------------------------------
16
17  --
18  -- Tabellenstruktur für Tabelle `document`
19  --
20
21  CREATE TABLE `document` (
22    `document_id` int(10) unsigned NOT NULL auto_increment,
23    `path` varchar(255) character set utf8 collate utf8_unicode_ci NOT NULL,
24    `filename` varchar(100) character set utf8 collate utf8_unicode_ci NOT NULL,
25    `hash` varchar(32) character set utf8 collate utf8_unicode_ci NOT NULL,
26    `token_count` mediumint(8) unsigned NOT NULL default '0',
```

```
27    PRIMARY KEY  (‘document_id‘)
28  ) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci AUTO_INCREMENT=8 ;
29
30  -- --------------------------------------------------------
31
32  --
33  -- Tabellenstruktur für Tabelle ‘new_token‘
34  --
35
36  CREATE TABLE ‘new_token‘ (
37    ‘document_id‘ int(10) unsigned NOT NULL,
38    ‘token‘ varchar(255) collate latin1_general_ci NOT NULL,
39    ‘position‘ mediumint(8) unsigned NOT NULL,
40    ‘field‘ varchar(255) collate latin1_general_ci NOT NULL
41  ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;
42
43  --
44  -- Triggers ‘new_token‘
45  --
46
47
48
49  -- --------------------------------------------------------
50
51  --
52  -- Tabellenstruktur für Tabelle ‘server‘
53  --
54
55  CREATE TABLE ‘server‘ (
56    ‘server_id‘ int(10) unsigned NOT NULL auto_increment,
57    ‘driver‘ varchar(255) character set utf8 collate utf8_unicode_ci NOT NULL,
58    ‘url‘ varchar(255) character set utf8 collate utf8_unicode_ci NOT NULL,
59    ‘user‘ varchar(50) character set utf8 collate utf8_unicode_ci NOT NULL,
60    ‘passwort‘ varchar(50) character set utf8 collate utf8_unicode_ci NOT NULL,
61    PRIMARY KEY  (‘server_id‘)
62  ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci AUTO_INCREMENT=1 ;
63
64  -- --------------------------------------------------------
65
66  --
67  -- Tabellenstruktur für Tabelle ‘token‘
```

```
--

CREATE TABLE `token` (
  `token_id` int(10) unsigned NOT NULL auto_increment,
  `token` varchar(255) character set utf8 NOT NULL,
  `server_id` int(10) unsigned NOT NULL default '0',
  `count` bigint(20) unsigned NOT NULL default '0',
  PRIMARY KEY  (`token_id`),
  UNIQUE KEY `token` (`token`)
) ENGINE=MyISAM  DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci AUTO_INCREMENT=9224

-- --------------------------------------------------------

--
-- Tabellenstruktur für Tabelle `token_document`
--

CREATE TABLE `token_document` (
  `token_id` int(10) unsigned NOT NULL,
  `document_id` int(10) unsigned NOT NULL,
  `position` mediumint(8) unsigned NOT NULL,
  `field` varchar(255) character set utf8 collate utf8_unicode_ci NOT NULL,
  KEY `token_id` (`token_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_general_ci;


DELIMITER //
CREATE TRIGGER `swp`.`addtoken` AFTER INSERT ON `swp`.`new_token`
 FOR EACH ROW BEGIN
  DECLARE new_token_id INT DEFAULT 0;

  SELECT token_id INTO new_token_id FROM token WHERE token = new.token;

  IF new_token_id = 0 THEN
      INSERT INTO token (token) VALUES (new.token);
      SET new_token_id = LAST_INSERT_ID();
  END IF;

  INSERT INTO token_document (token_id, document_id, position, field) VALUES
   (new_token_id, new.document_id, new.position, new.field);
  UPDATE document SET token_count = token_count + 1 WHERE document_id = new.document_
```

```
109     UPDATE token SET count = count + 1 WHERE token_id = new_token_id;
110
111     END//
112   DELIMITER ;
```